Marlin Protocol

Security Assessment

November 9th, 2020

For :
Marlin Protocol

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Marlin Protocol
Description	---
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 83001187a60aa69b7eeb10251b8728a6a4324579

Audit Summary

Delivery Date	November 9th, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	October 8th, 2020 - November 9th, 2020

Vulnerability Summary

Total Issues	45
Total Critical	0
Total Major	1
Total Medium	0
Total Minor	9
Total Informational	35

Executive Summary

This report represents the results of our engagement with the Marlin on a subset on their Marlin Protocol smart contracts.

Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the safety of the contract deployement.

Files In Scope

ID	Contract	Location
ARY	AddressRegistry.sol	contracts/stake-drop/AddressRegistry.sol
BLC	BridgeLogic.sol	contracts/Bridge/BridgeLogic.sol
DIS	Distribution.sol	contracts/stake-drop/Distribution.sol
SRY	StakeRegistry.sol	contracts/stake-drop/StakeRegistry.sol
SOE	StandardOracle.sol	contracts/stake-drop/StandardOracle.sol
TLC	TokenLogic.sol	contracts/Token/TokenLogic.sol
VRY	ValidatorRegistry.sol	contracts/stake-drop/ValidatorRegistry.sol
PLC	mPondLogic.sol	contracts/governance/mPondLogic.sol

Findings

ID	Title	Type	Severity	Resolved
ARY-01	Unlocked Compiler Version	Language Specific	Informational	✓
ARY-02	Visibility Specifiers Missing	Language Specific	Informational	✓
ARY-03	Bulk Address Addition Functionality	Volatile Code	Minor	✓
BLC-01	Unlocked Compiler Version	Language Specific	Informational	✓
BLC-02	Visibility Specifiers Missing	Language Specific	Informational	✓
BLC-03	Introduction of a `constant` Variable	Gas Optimization	Informational	✓
BLC-04	: Redundant Variable Initialization	Gas Optimization	Informational	✓
BLC-05	Redundant Mathematical Operation	Mathematical Operations	Informational	✓
BLC-06	Ambiguous Function	Volatile Code	Informational	✓
BLC-07	Function Optimization	Gas Optimization	Informational	✓
BLC-08	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
BLC-09	Ambiguous Error Message	Coding Style	Informational	✓
BLC-10	Statement Inconsistency	Gas Optimization	Informational	✓
BLC-11	Introduction of an `onlyOwner` modifier	Language Specific	Informational	✓
BLC-12	Unnecessary Use of `SafeMath`	Gas Optimization	Informational	✓
BLC-13	`storage` Over `memory`	Gas Optimization	Informational	✓
BLC-14	Potential Overflow	Volatile Code	Major	✓

ID	Title	Type	Severity	Resolved
BLC-15	Redundant `require` Statement	Gas Optimization	Informational	✓
BLC-16	Use of `SafeERC20.sol`	Volatile Code	Minor	✓
DIS-01	Unlocked Compiler Version	Language Specific	Informational	✓
DIS-02	Visibility Specifiers Missing	Language Specific	Informational	✓
DIS-03	Use of `SafeERC20.sol`	Volatile Code	Minor	✓
SRY-01	Unlocked Compiler Version	Language Specific	Informational	✓
SRY-02	Ambiguous Event	Gas Optimization	Informational	✓
SRY-03	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
SRY-04	Visibility Specifiers Missing	Language Specific	Informational	✓
SRY-05	Bulk Stake Addition Functionality	Volatile Code	Minor	✓
SOE-01	Unlocked Compiler Version	Language Specific	Informational	✓
SOE-02	Visibility Specifiers Missing	Language Specific	Informational	✓
SOE-03	Potential `source`-less Contact	Volatile Code	Minor	✓
SOE-04	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
TLC-01	Unlocked Compiler Version	Language Specific	Informational	✓
VRY-01	Unlocked Compiler Version	Language Specific	Informational	✓
VRY-02	Visibility Specifiers Missing	Language Specific	Informational	✓
VRY-03	Function Optimization	Gas Optimization	Informational	✓

ID	Title	Type	Severity	Resolved
VRY-04	Inexistent Input Sanitization	Volatile Code	Minor	✓
VRY-05	Ambiguous Implementation	Volatile Code	Minor	✓
VRY-06	Bulk Validator Addition Functionality	Volatile Code	Minor	✓
PLC-01	Unlocked Compiler Version	Language Specific	Informational	✓
PLC-02	Inefficient Data Type	Gas Optimization	Informational	✓
PLC-03	Introduction of an `onlyOwner` modifier	Language Specific	Informational	✓
PLC-04	Race Condition	Volatile Code	Minor	✓
PLC-05	Outdated Error Messages	Coding Style	Informational	✓
PLC-06	Partial Error Message	Coding Style	Informational	✓
PLC-07	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓

ARY-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	AddressRegistry.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.17`.

ARY-02: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	AddressRegistry.sol L10, L23-L24

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references, changed the visibility of the linked variables to `public` and removed the manual getter functions.

ARY-03: Bulk Address Addition Functionality

Type	Severity	Location
Volatile Code	Minor	AddressRegistry.sol L57-L59

Description:

The `addAddressBulk()` function should terminate early if one of the attempts to add a new address fails.

Recommendation:

We advise the team to add a `require` statement to check the result of the `addAddress()` invocation as an internal control mechanism. Also, returning the index of the failed address addition is a plus.

Alleviation:

The development team opted to consider our references, modified the `addAddress()` function to not return a boolean variable and changed the `addAddressBulk()` to directly call the `addAddress()` function, hence terminating early if the function breaks.

BLC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	BridgeLogic.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.17`.

BLC-02: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	BridgeLogic.sol L14-L28, L34-L35

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references, changed the visibility of the linked variables to `public` and removed the manual getter functions, while also removing some unnecessary state variables.

BLC-03: Introduction of a `constant` Variable

Type	Severity	Location
Gas Optimization	Informational	BridgeLogic.sol L17-L19, L22, L25-L28

Description:

The linked variables are assigned value during the initialization phase and do not get update again.

Recommendation:

We advise the team to change the mutability of the linked variables to `constant` and remove the `createConstants()` function.

Alleviation:

The development team opted to consider our references and changed the mutability of the variables that are not updated to `const`.

BLC-04: Redundant Variable Initialization

Type	Severity	Location
Gas Optimization	Informational	BridgeLogic.sol L55

Description:

When declaring variables without an initial value, they are assigned the specific data type's default value. Hence, the initialization of `uint256` to zero is redundant.

Recommendation:

We advise the team to remove the redundant assignments to the linked variables.

Alleviation:

The development team opted to consider our references and removed the linked variable, as it was deemed unnecessary.

BLC-05: Redundant Mathematical Operation

Type	Severity	Location
Mathematical Operations	Informational	BridgeLogic.sol L77

Description:

Redundant addition, as constant variable `startEpoch` is equal to zero.

Recommendation:

We advise the team to remove redundant code.

Alleviation:

The development team opted to consider our references and changed the mathematical operation.

BLC-06: Ambiguous Function

Type	Severity	Location
Volatile Code	Informational	BridgeLogic.sol L80-L82

Description:

The internal function `lockTimeEpoch()` is only used with two constant variables.

Recommendation:

We advise the team to revise the linked function.

Alleviation:

The development team opted to consider our references and replaced the `lockTimeEpoch()` function with the `lockTimeEpochs` constant variable.

BLC-07: Function Optimization

Type	Severity	Location
Gas Optimization	Informational	BridgeLogic.sol L94-L101, L103-L120

Description:

The linked functions can be further optimized.

Recommendation:

We advise the team to remove `else` block and directly use return after the `if` block.

Alleviation:

The development team opted to consider our references and changed the respective functions.

BLC-08: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	BridgeLogic.sol L131, L158

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The development team opted to consider our references and changed the linked comparisons with inequality ones.

BLC-09: Ambiguous Error Message

Type	Severity	Location
Coding Style	Informational	BridgeLogic.sol L138

Description:

The error is not fully covering the cases that the `require` statement checks.

Recommendation:

We advise the team to change the error message of the `require` statement to:
"total unlock amount should be less than or equal to requests_amount*effective_liquidity".

Alleviation:

The development team opted to consider our references and changed to a more descriptive error message.

BLC-10: Statement Inconsistency

Type	Severity	Location
Gas Optimization	Informational	BridgeLogic.sol L170

Description:

The value of `epoch + lockTimeEpoch(lockTime)` is stored in `memory`.

Recommendation:

We advise the team to use `_req.releaseEpoch` over `epoch + lockTimeEpoch(lockTime)`.

Alleviation:

The development team opted to consider our references and used the available `_req.releaseEpoch` variable.

BLC-11: Introduction of an `onlyOwner` modifier

Type	Severity	Location
Language Specific	Informational	BridgeLogic.sol L185-L187, L199-L202

Description:

The linked functions restrict the access control to the owner of contract.

Recommendation:

We advise the team to implement an `onlyOwner` modifier or introduce the `Ownable.sol` contract from OpenZeppelin.

Alleviation:

The development team opted to consider our references and implemented an `onlyOwner` modifier.

BLC-12: Unnecessary Use of `SafeMath`

Type	Severity	Location
Gas Optimization	Informational	BridgeLogic.sol L81

Description:

The `lockTimeEpoch()` function is unnecessarily using SafeMath's division function, as this calculation will never overflow/underflow.

Recommendation:

We advise the team to remove unnecessary code.

Alleviation:

The development team opted to consider our references and replaced the `lockTimeEpoch()` function with the `lockTimeEpochs` constant variable.

BLC-13: `storage` Over `memory`

Type	Severity	Location
Gas Optimization	Informational	BridgeLogic.sol L108

Description:

The linked variable should be stored to `storage` to save more gas.

Recommendation:

We advise the team to use `storage` over `memory` for the linked assignment.

Alleviation:

The development team opted to replace the complete `Requests struct` instance with the necessary `struct` member `amount`.

BLC-14: Potential Overflow

Type	Severity	Location
Volatile Code	Major	BridgeLogic.sol L133

Description:

The `totalUnlockableAmount` variable can overflow, as raw addition for the variable assignment provides zero protection to this vulnerability.

Recommendation:

We advise the team to use SafeMath's addition function (`add` invocation) to properly guard against a potential overflow.

Alleviation:

The development team opted to consider our references and used the `add` function exposed by the `SafeMath` library.

BLC-15: Redundant `require` Statement

Type	Severity	Location
Gas Optimization	Informational	BridgeLogic.sol L157-L160

Description:

The linked `require` statement redundantly checks whether the balance of the `msg.sender` is greater than zero or not, as the `require` statements in lines 152 and 161-164 guarantee that this conditional will always be true.

Recommendation:

We advise the team to remove redundant `require` statements.

Alleviation:

The development team opted to consider our references and removed the unnecessary `require` statement.

BLC-16: Use of `SafeERC20.sol`

Type	Severity	Location
Volatile Code	Minor	BridgeLogic.sol L189-L190, L203-L204, L214-L215

Description:

The linked `transfer` and `transferFrom` function call should be replaced with `safeTransfer` and `safeTransferFrom` invocations respectively.

Recommendation:

We advise the team to use the `SafeERC20` library when dealing with token transfers.

Alleviation:

The development team opted to consider our references and used the `SafeERC20` contract for the `TokenLogic` instances but not for `mPondLogic` ones, as it does fully match the `IERC20` inteface.

DIS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Distribution.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.17`.

DIS-02: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	Distribution.sol L11-L18

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references, changed the visibility of the linked variables to `public` and removed the manual getter functions, while also removing some unnecessary state variables.

DIS-03: Use of `SafeERC20.sol`

Type	Severity	Location
Volatile Code	Minor	Distribution.sol L42, L48, L71

Description:

The linked `transfer` and `transferFrom` function call should be replaced with `safeTransfer` and `safeTransferFrom` invocations respectively.

Recommendation:

We advise the team to use the `SafeERC20` library when dealing with token transfers.

Alleviation:

The development team acknowledged our exhibit, yet did not change the codebase, as the `mPondLogic` instances, as it does fully match the `IERC20` inteface.

SRY-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	StakeRegistry.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.17`.

SRY-02: Ambiguous Event

Type	Severity	Location
Gas Optimization	Informational	StakeRegistry.sol L32-L37

Description:

The event `StakeSkipped` seems to be redundant.

Recommendation:

We advise the team to remove the linekd event and to change the `if` conditional on L95 to a `require` while also removing the `else` block.

Alleviation:

The development team opted to consider our references, removed the `StakeSkipped` event and replaced the `if` conditional with a `require` statement.

SRY-03: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	StakeRegistry.sol L78

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The development team opted to consider our references and changed the linked comparisons with inequality ones.

SRY-04: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	StakeRegistry.sol L11-L16

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references, changed the visibility of the linked variables to `public` and removed the manual getter functions, while also removing some unnecessary state variables.

SRY-05: Bulk Stake Addition Functionality

Type	Severity	Location
Volatile Code	Minor	StakeRegistry.sol L116-L137

Description:

The `addStakeBulk()` function should terminate early if one of the attempts to add a new stake fails.

Recommendation:

We advise the team to return the index of the failed stake addition.

Alleviation:

The development team opted to consider our references, modified the `addStake()` function to not return a boolean variable and changed the `addStakeBulk()` to directly call the `addStake()` function, hence terminating early if the function breaks.

SOE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	StandardOracle.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.17`.

SOE-02: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	StandardOracle.sol L5

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references and changed the visibility of the linked variable to `public`.

SOE-03: Potential `source`-less Contact

Type	Severity	Location
Volatile Code	Minor	StandardOracle.sol L21-L24

Description:

if the function `renounceSource()` is called before `addSource()` one after deployment, then all functions of the codebase decorated with the `onlySource` modifier will be rendered uncallable.

Recommendation:

We advise the team to revise the respective code blocks.

Alleviation:

The development team opted to consider our references and modified the contract to ensure that there is at least one `source` present.

SOE-04: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	StandardOracle.sol L31

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The development team opted to consider our references and changed the linked comparison with inequality one.

TLC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	TokenLogic.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.17`.

VRY-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ValidatorRegistry.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.17`.

VRY-02: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	ValidatorRegistry.sol L8-L10

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references and changed the visibility of the linked variables to `public`.

VRY-03: Function Optimization

Type	Severity	Location
Gas Optimization	Informational	ValidatorRegistry.sol L16-L21

Description:

The `isFrozen()` function can be further optimized.

Recommendation:

We advise the team to change the linked function to:

```
function isFrozen(uint256 _epoch) public view returns (bool) {
    return (freezeTime[_epoch] != 0);
}
```

Alleviation:

The development team acknowledged our exhibit, but opted to remove the linked function from the codebase.

VRY-04: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	Minor	ValidatorRegistry.sol L36-L48, L62-L71

Description:

The linked functions omit checks for the input values.

Recommendation:

We advise the team to add the following `require` statements to the linked functions:

```
require(_epoch != 0, "Error Message"); (preferably in the modifier)
require(_validatorAddress != 0x0, "Error Message");
```

Alleviation:

The development team opted to consider our references and added `require` statements in both the linked functions, as well as the `isEpochNotFrozen` modifier.

VRY-05: Ambiguous Implementation

Type	Severity	Location
Volatile Code	Minor	ValidatorRegistry.sol L73-L83

Description:

According to the implementation, a user cannot unfreeze an epoch or toggle between "Frozen" and "Unfrozen" states.

Recommendation:

We advise the team to revise the respective code blocks.

Alleviation:

The development team acknowledged our exhibit and stated that intentionally frozen epochs cannot be unfrozen.

VRY-06: Bulk Validator Addition Functionality

Type	Severity	Location
Volatile Code	Minor	ValidatorRegistry.sol L50-L60

Description:

The `addValidatorsBulk()` function should terminate early if one of the attempts to add a new validator fails.

Recommendation:

We advise the team to return the index of the failed validator addition.

Alleviation:

The development team opted to consider our references, modified the `addValidator()` function to not return a boolean variable and changed the `addValidatorsBulk()` to directly call the `addValidator()` function, hence terminating early if the function breaks.

PLC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	mPondLogic.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.17`.

PLC-02: Inefficient Data Type

Type	Severity	Location
Gas Optimization	Informational	mPondLogic.sol L35

Description:

The data type of the nested `mapping` is `uint32`. The mapping will automatically cast it to a `uint256` type before the look-up.

Recommendation:

We advise the team to change the `key` type of the nested `mapping` to `uint256`.

Alleviation:

The development team acknowledged our exhibit but opted not to change the referenced data type.

PLC-03: Introduction of an `onlyOwner` modifier

Type	Severity	Location
Language Specific	Informational	mPondLogic.sol L127, L133

Description:

The linked functions restrict the access control to the owner of contract.

Recommendation:

We advise the team to implement an `onlyOwner` modifier or introduce the `Ownable.sol` contract from OpenZeppelin.

Alleviation:

The development team opted to consider our references and implemented an `onlyOwner` modifier.

PLC-04: Race Condition

Type	Severity	Location
Volatile Code	Minor	mPondLogic.sol L162-L185

Description:

The contract suffers from the inhereted race condition of the ERC-20 standard that stems from its `approve()` function.

Recommendation:

We advise the team to only use the `approve()` function when the spender's allowance is zero. In any other case, the usage of `increaseAllowance()` and `decreaseAllowance()` functions will help mitigate this problem.

Alleviation:

The development team opted to consider our references and implemented the `increaseAllowance()` and `decreaseAllowance()` functions.

PLC-05: Outdated Error Messages

Type	Severity	Location
Coding Style	Informational	mPondLogic.sol L99, L178, L209, L235, L242, L300, L304, L306, L336, L340, L342, L371, L413, L418, L434, L439, L452, L456, L460, L466, L471, L477, L482, L503, L516, L531

Description:

The linked error messages point to the old "Comp" contract instead of the new "mPond".

Recommendation:

We advise the team to update the linked error messages.

Alleviation:

The development team opted to consider our references and updated the error messages of the linked `require` statements.

PLC-06: Partial Error Message

Type	Severity	Location
Coding Style	Informational	mPondLogic.sol L205, L229

Description:

The error message should also include the `enableAllTransfers()` functionality as well.

Recommendation:

We advise the team to update the linked error message.

Alleviation:

The development team opted to consider our references and updated the error messages of the linked `require` statements.

PLC-07: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	mPondLogic.sol L354, L494, L495, L510, L535

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The development team opted to consider our references and changed the linked comparisons with inequality ones.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.